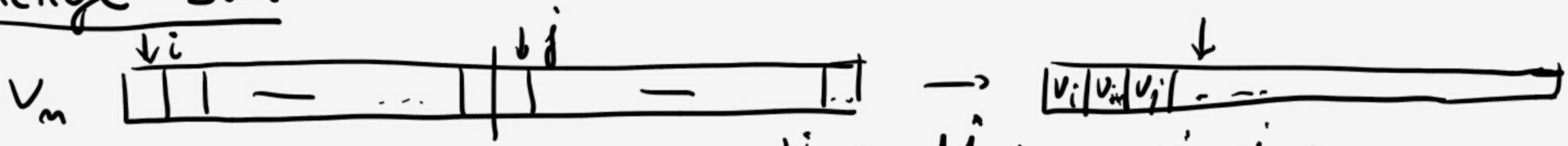
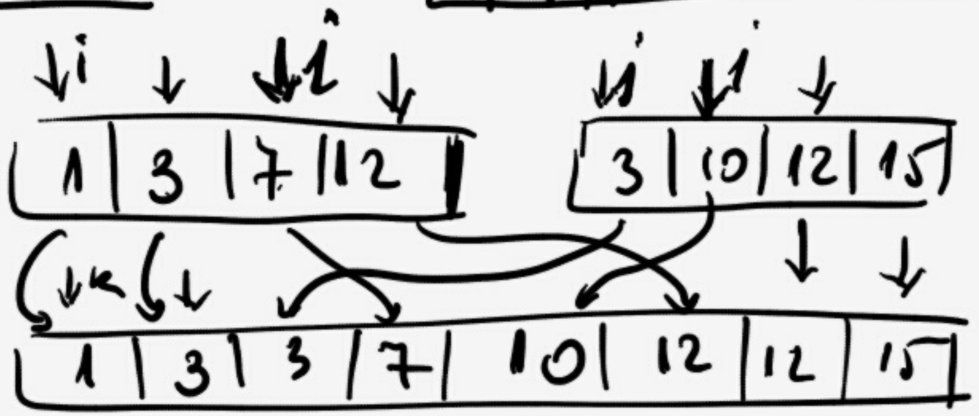
Merge Sort



1. Split v
2a. Sort - 0 → $\frac{n}{2}$
2b. Sort - $\frac{n}{2}$ - n
3. Merge



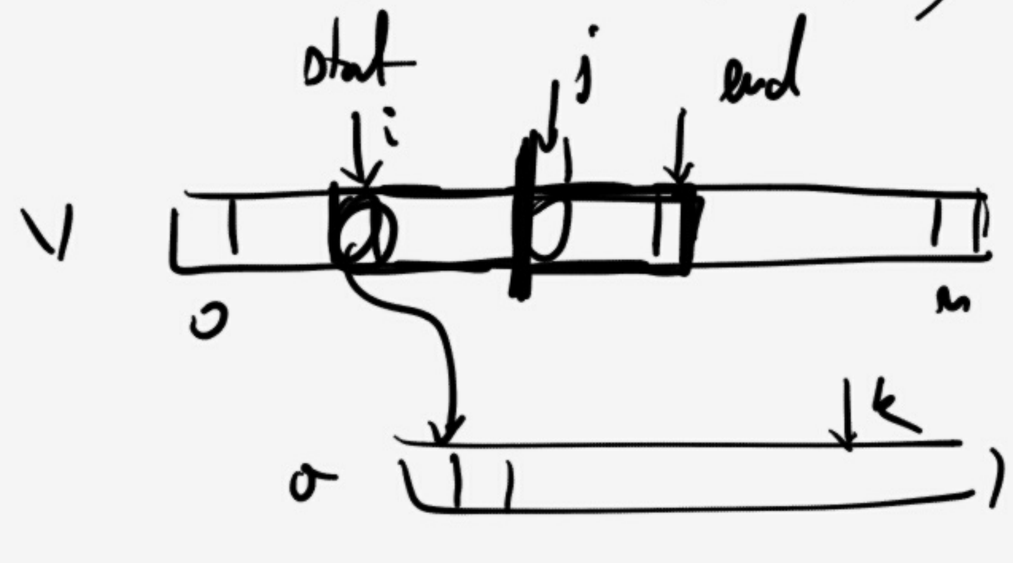
Merge Sort - Pseudocod

exclusiv → $O(n \lg_2 n)$

inclusiv

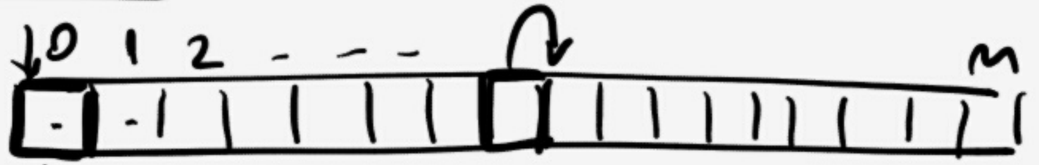
```
mSort(v, start, end)
    middle ← (start + end)/2
    if (start == end) return;
    mSort(v, start, middle)
    mSort(v, middle, end)
    alloc a[end - start]
    i ← start
    j ← middle
    for (k = 0 ... end - start - 1)
        if v[i] < v[j] || j == end
            a[k] ← v[i]
            i++
        else
            a[k] ← v[j]
            j++
    endfor
    v[start: end] ← a
```

v[n] - mSort(v, 0, n)



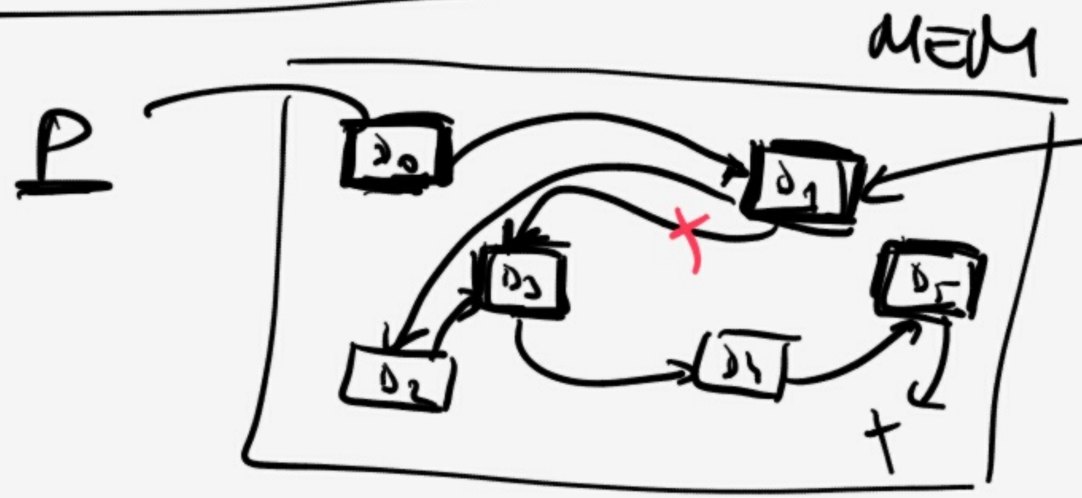
Liste (simplu înlănțuite)

Rep. Vector V



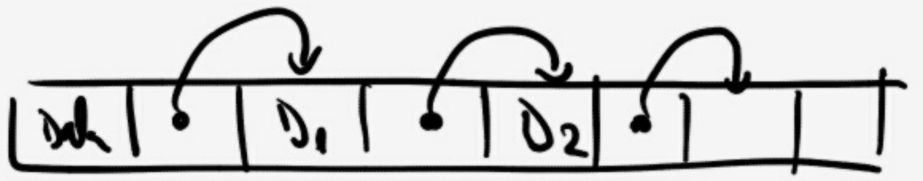
sizeof(data_type)

$V[k] \equiv V \pm k$ (* sizeof(data_type))



Inserție în O(1)
Accesul este secvențial în O(n)

nCells * sizeof(cell)
||
1+P

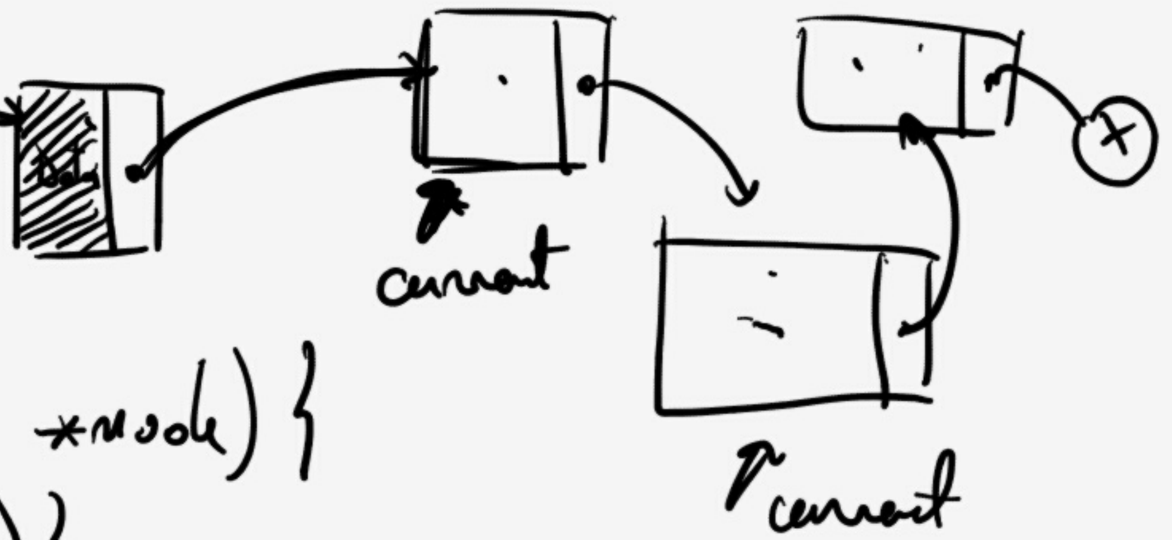



```
struct Node {
        int data;
        struct Node *next;
};
```

```
struct Node *myNode = (struct Node*)malloc(sizeof(struct Node));
myNode -> data = 743;
myNode -> next = NULL;
```

① Iterație recursivă.

struct Node *p;




```
void printList(struct Node *node) {
    if (node->next != NULL) {
        printf("%d\n", node->next->data);
        printList(node->next);
    }
}
```

```
void printListNonRecursive(struct Node *node) {
    struct Node *current;
    for (current = node->next; current != NULL;
         current = current->next) {
        printf("%d, ", current->data);
    }
}
```

```
unsig. char i, j;
    for(i=0, j=1; i<j; i++, j++){
    //...
    }
```

Căutare în listă → secvențial $O(n)$

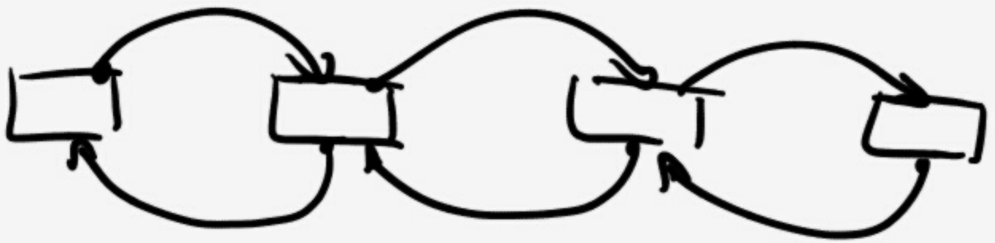
Operație	Vector	Listă simplu înlănțuită
Acces	$O(1)$	$O(n)$
Căutare	$O(n)$ / $O(\log_2 n)$	$O(n)$ / $O(n)$
inserție după k	$O(n)$ / $O(\log_2 n)$ + $O(n) = O(n)$	$O(n) + O(1) = O(n)$
inserție pe poz i	$O(n)$	$O(n) + O(1) = O(n)$

Sortare pt. liste simplu înlănțuite

- Bubble-sort ✓ $O(n^2)$
- Insertion Sort ✓ $o(n^2)$
- Quick Sort ✗
- Merge Sort ✓ $> o(n \log_2 n)$

LSÎ

LDÎ



Overhead double de memorie

```
struct DNode {
        int data;
        struct DNode *next;
        struct DNode *prev;
};
```